Requirement:

Goal: Develop a .NET Core / .NET 5+ component (e.g., a class library) that interacts

with a public API to fetch, process, and potentially cache data. This simulates

integrating with and managing data from an external provider, a common task when

extending larger platforms.

Scenario: Imagine your platform needs to display user information or configuration

data fetched from an external system. You need to build a reliable service component

to handle this interaction. We will use the public https://reqres.in/API as a stand-in for

this external system.

Core Requirements:

1. API Client Implementation:

a. Create a client service to interact with the following reqres.in endpoints:

■ GET https://reqres.in/api/users?page={pageNumber} (to get

paginated lists of users)

■ GET https://reqres.in/api/users/{userId} (to get details for a

single user)

b. Use HttpClient (preferably via IHttpClientFactory if integrated into a

host, or demonstrate proper HttpClient usage otherwise).

c. Implement using async/await correctly.

2. Data Modeling & Mapping:

a. Define internal C# models (POCOs/DTOs) to represent the relevant user

data returned by the API (e.g., id, email, first\_name, last\_name).

b. Map the JSON response from the API to your internal models.

3. Service Layer:

a. Create a service class (e.g., ExternalUserService) that encapsulates the

logic for fetching data.

b. Implement methods like:

■ Task<User> GetUserByIdAsync(int userId)

■ Task<IEnumerable<User>> GetAllUsersAsync() (Handle

pagination internally to fetch all users across available pages).

4. Configuration:

a. Make the base URL (https://reqres.in/api/) configurable (e.g., read from

appsettings.json if using a host, or demonstrate how it would be

configured).

5. Error Handling & Basic Resilience:

a. Handle potential errors gracefully:

■ API request failures (network issues, timeouts).

■ Non-success HTTP status codes (e.g., 404 Not Found when

requesting a specific user).

■ Deserialisation errors.

b. Return meaningful results or throw specific exceptions to indicate

different failure modes.

c. Consider (even if just commented or discussed in README) how you

might add basic retry logic for transient network errors.

Approach:

As per our requirement, by using the external public API, <https://reqres.in/API> we have to fetch the Users

1. Created a Class Library Project to call the public API.
2. Created Models based on the Public API Attributes. Users,UsersPage and UserWrapper

User- Created the entities for each User’s, Id,Email,FirstName,LastName and Avatar(Photograph)

UserPage- Indicates the Page Defination for each Page

UserWrapper- Indicates the data column that I have received the from the service and streamline to User.

1. Created Services folder and created an Interface “IExternalUserService ”to their Implementation in the “ExternalUserService” class.

IExternalUserService. GetUserByIdAsync (int userId)

This method is for getting the details of the user by the UserId. We are sending the userId as Parameter and retrieve the info

IExternalUserService. GetAllUsersAsync

This method is for getting the details of the users by the PageId. We are sending the PageID as Parameter and retrieve the info.

1. Build the Class Library.
2. Created the ConsoleApplication, Added the reference of Class Library to it.
3. In the Program file, calling the services and printed the results in console window.
4. Added the XUnit Test Project to the solution
5. Wrote the Test case to call the sevice GetUserById , and give the fact Data as for 2 and Call the service by passing parameter as 2 and by using the Assert.Equal matching the value.